15-451/651 Algorithms, Fall 2017

Homework #3 **Due: October 10, 2017**

This HW has three regular problems, and one programming problem. All problems on written HWs are to be done *individually*, no collaboration is allowed.

Solutions to the three written problems should be submitted as a single PDF file using `gradescope`, with the answer to each problem starting on a new page.

Submission instructions for the programming problem will be posted on the website and Piazza.

(25 pts) 1. **2D Matrix Match.**

Suppose you are given a $n \times m$ matrix T and a $n' \times m'$ matrix P ($n' \leq n$ and $m' \leq m$). All entries of T and P are 0 or 1.

We say P *occurs* at position (i, j) of T if the submatrix $T[i \ldots i+n'-1, j \ldots, j+m'-1] = P$.

Let

$$h_p(X) = \sum_{i=0}^{r(X)-1} \sum_{j=0}^{c(X)-1} X_{ij} 2^{i \cdot c(X)+j} \mod p$$

where p is a prime, and $r(X)$ and $c(X)$ are, respectively, the number of rows and columns in a matrix $X = (X_{ij})$.

(a) Give an $O(mn)$ algorithm to compute $h_p(S)$ for all S, where where S is a submatrix of T whose dimensions are that of P. (Assume that you can do modular arithmetic with integers of value $O(p)$ in constant time.)

(b) Consider the following algorithm to find all occurrences of P in T. Choose p to be a random prime not exceeding M. Using part (a) compute $h_p(S)$ for all relevant submatrices S of T. Output those for which $h_p(S) = h_p(P)$.

Give the best upper bound you can for the probability that your algorithm will return a false positive match. Your answer will be a funciton of n, m, and M.

(25 pts) 2. **Swapping Chaining**

Instead of creating a linked list for each hash bucket, suppose we resolve collisions in a different way.

We maintain two hash functions h_1 and h_2, and a table T, each entry of which can be empty, or store a (key, data) pair. We use the following algorithms to access and update the hash table:

```
Lookup(x):
    if T[h1(x)].key == x, return T[h1(x)].data
    if T[h2(x)].key == x, return T[h2(x)].data
    return NOT_FOUND

Insert (x, data):
    while true:
        if T[h1(x)] is empty: { T[h1(x)] = (x, data); return }
        swap T[h1(x)] and (x, data)
        if T[h2(x)] is empty: { T[h2(x)] = (x, data); return }
        swap T[h2(x)] and (x, data)
```

Note that Insert could get into an infinite loop! This question starts to analyze when that can happen. To do that, it uses the following class of graphs $G(R) = (V, E(R))$, where R is a set of keys currently stored in the table.

$$V = \text{slots in } T$$
$$E(R) = \{\{h_1(x), h_2(x)\} \text{ for each } x \text{ in } R\}$$

This graph may have self-loops and multi-edges.

Let R be the set of items in the table just before we call Insert(x).

(a) Show that Insert(x) always fails if the connected component of $G(R \cup \{x\})$ that contains the edge for x contains two or more cycles (a self-loop counts as a cycle).

(b) Show that Insert(x) terminates (eventually) if the connected component of $G(R \cup \{x\})$ that contains the edge for x contains at most one cycle.

(c) Let c be the number of nodes in the connected component of $G(R \cup \{x\})$ that contains the edge for x. Show that if Insert(x) terminates, it does so after at most 2c swaps.

(Unfortunately, it is beyond the scope of a homework to show that if you choose good hash functions then the chance that (a) happens is small, and that the expected size of the connected components is small. But in fact, you can show both, making this actually an attractive scheme for a hash table!)

(25 pts) 3. **How Universal?**

In this problem we will modify the matrix method to construct a hash family that you will analyze for L-universality.

As in lecture, let the keys be u-bits long, and let the table size be $M = 2^m$. So an index is m-bits long. This time we pick two random $m \times u$ 0/1 matrices called A_1 and A_2. Our hash function is defined as follows:

$$h(x) = A_1 x \oplus A_2 \bar{x}.$$

The matrix-vector product uses modulo 2 arithmetic. The $\oplus$ indicates the exclusive or of two vectors of length m, and the symbol $\bar{x}$ indicates the bitwise complement of x. This defines a hash family H.

(a) Prove that H is 3-Universal.

(b) Prove that H is not 4-Universal.

(25 pts) 4. **Double String** Write a program which takes as input a string of lower case letters (from among "`a`" ... "`z`"), and determines if there exists somewhere in the string a pattern of the same substring twice in a row. For example, if the input string is "`eatmorecouscousyummy`" the output could be "`start = 7 length = 4`". If there are several solutions your program can output any of them. (An alternative solution to the above intput is "`start = 17 length = 1`". On the other hand if there is no such pattern, then the program should output "`no double`". The time limit is 10 seconds.

Details on how to submit, grading policy, etc., will be on the course website and piazza soon.

INPUT: A string of lower case alphabetical letters of length at most 50000.

OUTPUT: As described above, and in the examples below.

For example, if the input is:

```
aa
```

then the output would be

```
start = 0  length = 1
```

Or if the input is:

```
abracadabra
```

then the output would be

```
no double
```